Nanodegree Engenheiro de Machine Learning

Proposta de projeto final - Pokémon batalha - Usando Random Forest (Supervisionado)

Paulo Cotta
19 de Agosto de 2018

I. Definição

A proposta do trabalho é executar processos onde os pokémons possam batalhar entre rodadas, sendo que cada treinador possa escolher apenas um pokémon.

A ideia é saber em que cada rodada qual Pokémon possa vencer.

O DataSet escolhido foi retirado da plataforma Kaggle, conforme link a seguir: https://www.kaggle.com/terminus7/pokemon-challenge/kernels.

Segue uma introdução sobre o local que será utilizado:

> Kanto (em japonês: カント一地方, Kantō-chihō) é um país ou região fictícia, da série Pokémon. Sua geografia é baseada na região de Kanto, uma região da ilha de Honshu, no Japão, de onde vem seu nome. A semelhança entre as formações de baía vistas no mapa do jogo e as formações reais de Sagami Bay, Suruga Bay e a Baía de Tokyo é particularmente impressionante.
> Kanto localiza-se a leste de Johto; presumivelmente, eles formam um pequeno continente.
> Fonte: Wikipédia: https://pt.wikipedia.org/wiki/Kanto_(Pok%C3%A9mon).

Visão geral do projeto

Pokémon é uma série de jogos eletrônicos desenvolvidos pela Game Freak e publicados pela Nintendo como parte da franquia de mídia Pokémon. Lançado pela primeira vez em 1996 no Japão para o console Game Boy, a principal série de jogos de RPGs, que continuou em cada geração em portáteis da Nintendo. Fonte: https://pt.wikipedia.org/wiki/Pok%C3%A9mon_(s%C3%A9rie_de_jogos_eletr%C3%B4nicos) - Acessado: 20/09/2018

A ideia do projeto é simular batalhas onde possamos brincar como se estivessemos em um processo de jogar o jogo Pokémon.

Dessa forma foi usado um .csv que foi disponibilizado pela plataforma Kaggle e assim conseguimos atingir os objetivos e simular uma batalha. Abaixo segue imagem representativa de alguns Pokémons.

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	4	Mega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	5	Charmander	Fire	NaN	39	52	43	60	50	65	1	False

Agora o problema em nossa mão é que, dado alguns recursos sobre cada Pokémon, como seu ataque, defesa ou seu valor de velocidade, etc, precisamos prever o vencedor de uma batalha Pokémon aleatória que nunca ocorreu antes.

Um conjunto de dados é a coleta de grande quantidade de dados sobre um tópico específico. Este conjunto de dados nos fornece informações como os Pontos de Vida, ataque, Defesa Especial e tempo em que o Pokémon é lendário **TRUE** ou não **FALSE**. A tabela acima mostra os dados dos primeiros 5 Pokémon, mas há um total de 800 Pokémon (significa 800 linhas) no conjunto de dados.

Descrição do problema

O problema que será resolvido, será de poder criar um simulador de batalhas e dessa forma ter as saídas dos ganhadores de forma que a máquina entendeu, obteve um aprendizado e executou de forma autônoma.

A proposta do trabalho que era executar todo o jogo e começar a predizer de forma autônoma, não ocorreu por conta do prazo, por conta de múltiplas variáveis que poderiam exercer o potencial de máquina, então dessa forma foi decidido reduzir o trabalho apenas as batalhas entre os Pokémons.

Para este problema será usado Random Forest para a aplicação, pois percebi que posso usar várias aplicações no algoritmo para efetuar uma decisão.

Métricas

A ideia é que possa ser utilizado um algoritmo que possa decidir de forma rápida, ágil e consiga exercer suas decisões de forma rápida. Em algumas pesquisas, resolvi escolher o Random Forest para efetuar suas decisões de forma consistente e chegar em uma melhor acurácia.

Dizendo de modo simples: o algoritmo de florestas aleatórias cria várias árvores de decisão e as combina para obter uma predição com maior acurácia e mais estável.

Links de pesquisa de maior relevância para a pesquisa:

- https://www.ufrgs.br/sbai17/papers/paper_98.pdf
- http://wiki.dpi.inpe.br/lib/exe/fetch.php?media=ser300:alunos2014:cesare_monografia.pdf
- https://medium.com/machina-sapiens/o-algoritmo-da-floresta-aleat%C3%B3ria-3545f6babdf8

```
# Saída da Acurácia do treino

print('Versão do SKLearn {}.'.format(sklearn.__version__))

clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(x_train, y_train)
pred = model.predict(x_test)
print('Acurácia de = ', accuracy_score(pred, y_test)*100) # Teste inicial 65% de acurácia
```

```
Versão do SKLearn 0.20rc1.
Acurácia de =  65.24
```

Suponha que temos uma tarefa de prever o animal com base nas características como tipo, altura, peso ou velocidade do animal. Esta tarefa pode ser facilmente modelada usando uma árvore de decisão. Então, em cada ponto da árvore de decisão, fazemos uma pergunta e, dependendo da resposta, dividimos a árvore em sub-árvores. Este processo é repetido até prevermos um animal. Assim, dado um conjunto de dados, um Classificador de Árvore de Decisão fará as perguntas certas (aumentando o Ganho de Informações) em cada ponto, de modo a dividir a árvore de forma a aumentar a confiança para cada previsão (aumentando a pureza do resultado).

Como as florestas são uma coleção de árvores, o Random Forest Classifier usa várias árvores de decisão e, finalmente, combina os resultados de cada árvore de decisão para prever seu resultado final.

Por fim, foi criado um classificador de floresta aleatório da seguinte maneira. Os n_estimators fornecem o Número de árvores de decisão valor=100 usados para fazer a floresta.

II. Análise

A ideia da análise é obter assertividade sobre o problema, demonstrar e provar que é possível executar.

Exploração dos dados

Conforme apresentado anteriormente foi utilizado DataSet referente aos Pokémons e suas batalhas. Abaixo segue uma imagem com os campos trabalhados.

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	4	Mega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	5	Charmander	Fire	NaN	39	52	43	60	50	65	1	False

Conforme detalhado o DataSet possui:

- Name: O nome da especie do Pokémon;
- Type 1: O tipo natural do Pokémon, o primeiro nível hierárquico da sua espécie;
- Type 2: A sua segunda forma hierárquica da sua espécie e muitas vezes não interfere na decisão;
- HP: Pontos de vida do Pokémon;
- Attack: O total de ataque para aquela espécie;
- Defense: O total de defesa que aquela espécie Pokémon possui;
- Sp. ATK: O total de ataques especiais para aquele Pokémon em especifico;
- Sp. DEF: O total de defesa especial para aquele Pokémon em especifico;
- Speed: O total de velocidade que um Pokémon possui;
- Generation: É qual geração aquele Pokémon aparece nos jogos e nas séries;
- Legendary: No mundo Pokémon existem alguns Pokémons lendários e neste campo define qual Pokémon está nessa categoria.

Dessa forma foi revisto as variáveis de entrada e decidido executar algumas normalizações, exemplo retirada dos NaNs, transformar o booleano True e False para 0 e 1.

```
RangeIndex: 800 entries, 0 to 799
Data columns (total 12 columns):
#             800 non-null int64
Name          799 non-null object
Type 1        800 non-null object
Type 2        414 non-null object
HP            800 non-null int64
Attack        800 non-null int64
Defense       800 non-null int64
Sp. Atk       800 non-null int64
Sp. Def       800 non-null int64
Speed         800 non-null int64
Generation    800 non-null int64
Legendary     800 non-null bool
dtypes: bool(1), int64(8), object(3)
memory usage: 69.6+ KB
```

Exemplo Type 2 que possuímos 386 NaNs no DataSet.

Visualização exploratória

Foi decidido que boa parte das variáveis do DataSet são importantes para o trabalho, exemplo a imagem abaixo:

```
########################################
# Processo de normalização dos dados a partir das colunas definidas
########################################
def normalization(data_df):
    stats=["HP","Attack","Defense","Sp. Atk","Sp. Def","Speed","Legendary"]
```

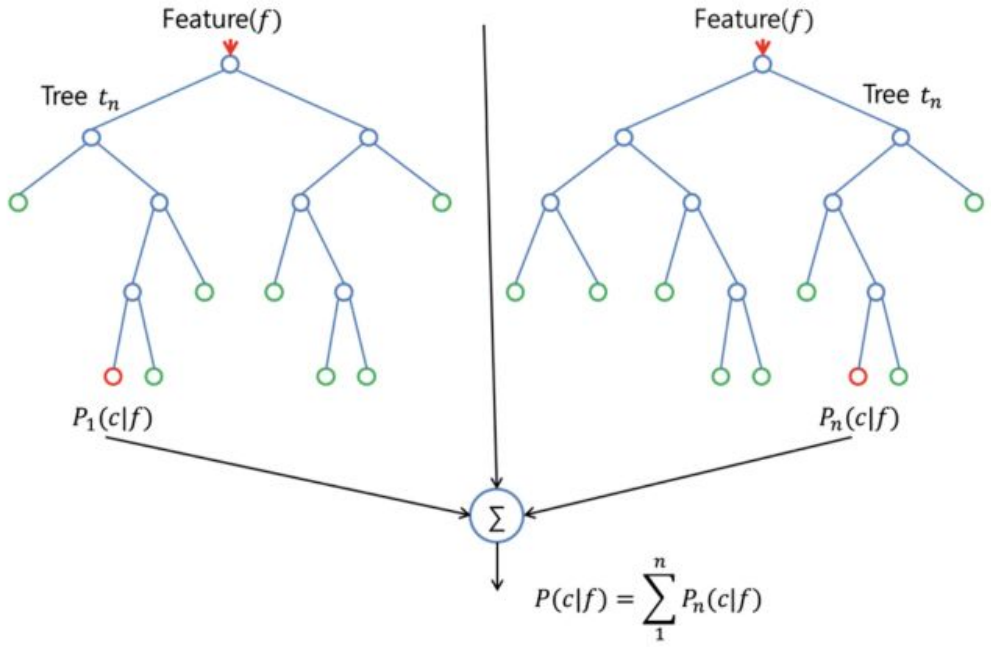
Porque utilizar as sete variáveis? Elas são de extrema relevância para que possa utilizar o HP para subtrair no processo de batalha, ataque para disparar o ataque, defesa para defender quando precisar e evitar a subtração dos dados, velocidade do ataque, da defesa e do disparo e se é lendário que conta muito para obter uma melhor qualificação nos seus ataques.

Algoritmos e técnicas

O algoritmo selecionado foi o Random Forest para aplicar as decisões do Pokémon vencedor. Abaixo segue os motivos e as razões de escolher o modelo de aplicação de Machine Learning.

- O processo do Random Forest é supervisionado e cria um processo chamado ensemble onde cria o processo de árvore aleatória de decisão;

- Bagging: A idéia principal do método de bagging é que a combinação dos modelos de aprendizado aumenta o resultado geral;
- Uma grande vantagem do algoritmo de florestas aleatórias é que ele pode ser utilizado tanto para tarefas de classificação quanto para regressão, o que representa a maioria dos sistemas de aprendizagem de máquina atuais;
- Com raras exceções um classificador de floresta aleatória tem todos os hiperparâmetros de uma árvore de decisão e também todos os hiperparâmetros de um classificador de bagging, para controlar a combinação de árvores;
- Portanto, quando você está criando uma árvore no floresta aleatória, apenas um subconjunto aleatório das características é considerada na partição de um nodo.



Outra grande qualidade das florestas aleatórias é a facilidade para se medir a importância relativa de cada característica (feature) para a predição. Sklearn provê uma excelente ferramenta para isto, que mede a importância das características analisando quantos nodos das árvores, que usam uma dada característica, reduzem impureza geral da floresta. Ele calcula este valor automaticamente para cada característica após o treinamento e normaliza os resultados para que a soma de todas as importâncias seja igual a 1.

Suponha que temos uma tarefa de prever o animal com base nas características como tipo, altura, peso ou velocidade do animal. Esta tarefa pode ser facilmente modelada usando uma árvore de decisão. Então, em cada ponto da árvore de decisão, fazemos uma pergunta e, dependendo da resposta, dividimos a árvore em sub-árvores. Este processo é repetido até prevermos um animal. Assim, dado um conjunto de dados, um Classificador de Árvore de Decisão fará as perguntas certas (aumentando o Ganho de Informações) em cada ponto, de modo a dividir a árvore de forma a aumentar a confiança para cada previsão (aumentando a pureza do resultado).

Como as florestas são uma coleção de árvores, o Random Forest Classifier usa várias árvores de decisão e, finalmente, combina os resultados de cada árvore de decisão para prever seu resultado final.

Por fim, foi criado um classificador de floresta aleatório da seguinte maneira. Os n_estimators fornecem o Número de árvores de decisão valor=100 usados para fazer a floresta.

Benchmark

Conforme mostrado o processo de cada variável ela interfere no valor da batalha de cada Pokémon, momento em que escolhe Pokémon 1 VS Pokémon 2 e assim por diante. Foi escolhido 100 iterações de árvores de decisão para o processo de aprendizado.

Reiterando a tarefa, dois Pokémon com seu conjunto de recursos (velocidade, ataque, etc) qual deles vai ganhar.

O treinamento do classificador no conjunto de dados de Pokémon (ou seja, x_train) e minimizamos a perda entre os valores previstos e reais (y_train) no conjunto de treinamento. Treinar aqui significa encontrar as relações entre os diferentes recursos do conjunto de dados para fazer previsões.

Em seguida, calculamos a precisão de nosso classificador, que é de 65% (significa que nosso classificador irá prever resultados corretos para 95 de 100 partidas), o que é uma boa precisão para começar.

```
# Saída da Acurácia do treino

print('Versão do SKLearn {}.'.format(sklearn.__version__))

clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(x_train, y_train)
pred = model.predict(x_test)
print('Acurácia de = ', accuracy_score(pred, y_test)*100) # Teste inicial 65% de acurácia
```

```
Versão do SKLearn 0.20rc1.
Acurácia de =  65.24
```

Até agora, concluímos todas as etapas necessárias desde a criação de um classificador até o treinamento e agora será testado efetuando uma batalha Pokémon.

III. Metodologia

Processo de explicação entre o algoritmo e a proposta a ser executada.

Pré-processamento de dados

Conforme colocado anteriormente, o processo de pré processar o DataSet de pokemon.csv e combats.csv foi importante para evitar problemas de identificação ou *phantom line* nas saídas das predições.

O processo selecionado foi limpar os NaNs, retirar os itens que eram a palavra booleana, exemplo True ou False e transformar para 1 e 0 respectivamente.

Dessa forma evita-se problemas de encontrar um NaN ou *phantom line* e obter uma saída não esperada ou executar um erro de float32 bits no método, porque não foi possível considerar um item nulo.

Implementação

A implementação consiste em leitura dos dados, tratamento dos dados, concatenar os dados de Pokémon e Combate, executar o treino, executar o processo de decisão do Random Forest e executar uma predição exata do combate dos Pokémons.

Outra grande qualidade das florestas aleatórias é a facilidade para se medir a importância relativa de cada característica (feature) para a predição. Sklearn provê uma excelente ferramenta para isto, que mede a importância das características analisando quantos nodos das árvores, que usam uma dada característica, reduzem impureza geral da floresta. Ele calcula este valor automaticamente para cada característica após

o treinamento e normaliza os resultados para que a soma de todas as importâncias seja igual a 1.

Se você não sabe como funciona uma árvore de decisão e se você não sabe o que é uma folha ou nodo, em uma árvore de decisão cada nodo interno representa um teste em um atributo (exemplo: se ao jogar uma moeda e uma das opções for cara ou coroa), cada ramo da árvore representa uma saída do teste, e cada folha representa um rótulo de classe (decisão tomada após calcular todos os atributos). Um nodo que não tem descendentes é uma folha.

Como mencionado anteriormente, Random Forest é uma coleção de árvores de decisão, mas há algumas diferenças entre uma árvore 100% de decisão e uma floresta randômica.

Foi treinado uma árvore com um DataSet de treinamento e rótulos, foi elaborado um conjunto de regras que serão utilizadas para realizar predições.

Processo de limpeza para clarear os nodos da folha do algoritmo Random Florest:

```
final_data.info()

final_data['HP'] = final_data['HP'].fillna(0)
final_data['Attack'] = final_data['Attack'].fillna(0)
final_data['Defense'] = final_data['Defense'].fillna(0)
final_data['Sp. Atk'] = final_data['Sp. Atk'].fillna(0)
final_data['Sp. Def'] = final_data['Sp. Def'].fillna(0)
final_data['Speed'] = final_data['Speed'].fillna(0)
final_data['Legendary'] = final_data['Legendary'].fillna(0)
```

Através da inspeção da importância das características, pode-se decidir quais características deixar de fora do modelo, já que eles não contribuem o suficiente ou nada para o processo de predição. Isto é importante, porque uma regra geral em aprendizagem de máquina é que quanto mais características você tem, mais provavelmente seu modelo irá sofrer de sobreajuste (overfitting) e vice versa.

Refinamento

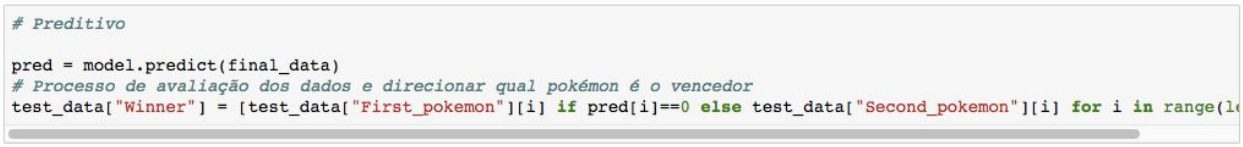
Conforme apresentado anteriormente na imagem:

```
final_data.info()

final_data['HP'] = final_data['HP'].fillna(0)
final_data['Attack'] = final_data['Attack'].fillna(0)
final_data['Defense'] = final_data['Defense'].fillna(0)
final_data['Sp. Atk'] = final_data['Sp. Atk'].fillna(0)
final_data['Sp. Def'] = final_data['Sp. Def'].fillna(0)
final_data['Speed'] = final_data['Speed'].fillna(0)
final_data['Legendary'] = final_data['Legendary'].fillna(0)
```

Esse processo de refinamento e limpeza ajudou em uma melhor saída e escolha da simulação da batalha Pokémon.

Após isso foi utilizado a técnica de predizer quem ou qual Pokémon irá ganhar a batalha, exemplo na imagem abaixo:



As duas colunas correspondem aos Pokémon que vão competir. Será alimentado esses dois Pokémon no classificador e ele retornará o vencedor mais provável para essa batalha. Lembre-se de que o classificador não está apenas predizendo aleatoriamente o vencedor. Na verdade, ele está analisando vários parâmetros cuidadosamente para chegar à decisão correta.

Primeiramente, há o hiperparâmetro n_estimators, que indica o número de árvores construídas pelo algoritmo antes de tomar uma votação ou fazer uma média de predições. Em geral, uma quantidade elevada de árvores aumenta a performance e torna as predições mais estáveis, mas também torna a computação mais lenta.

Outro hiperparâmetro importante é max_features, que indica o número máximo de características a serem utilizadas pelo Floresta Aleatória na construção de uma dada árvore.

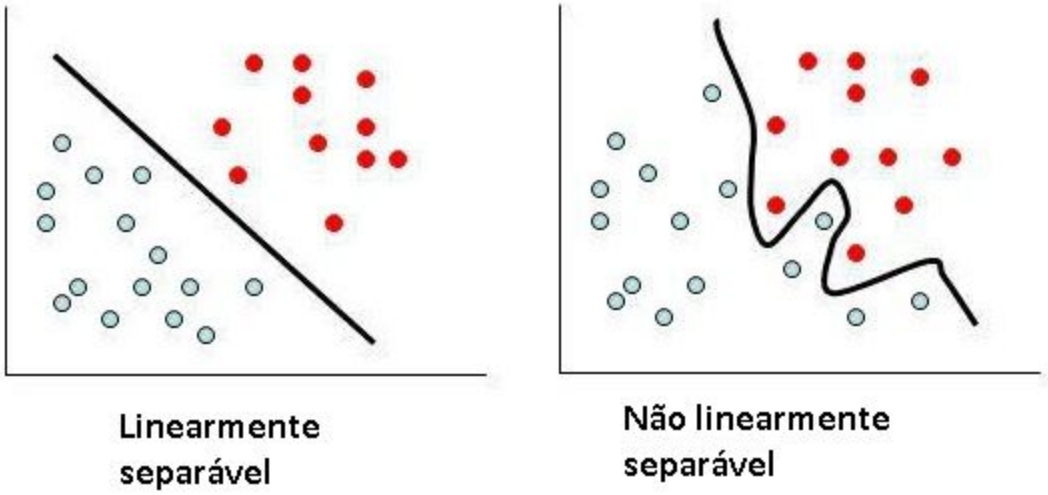
O algoritmo Floresta Aleatória é utilizado em muitas áreas diferentes, tal como setor bancário, mercado financeiro, medicina, comércio eletrônico. No setor bancário ele é utilizado, por exemplo, para detectar clientes que irão utilizar os serviços bancários mais frequentemente que outros e pagar suas dívidas pontualmente. Neste domínio ele também é utilizado para detectar fraudes de clientes que querem lesar o banco. No setor financeiro, ele é utilizado para determinar o desempenho futuro de uma ação. Na área de saúde ele é utilizado para identificar a correta combinação de componentes em medicina, e também é utilizado para analisar o histórico médico de um paciente para identificar doenças.

IV. Resultados

Modelo de avaliação e validação

Meu modelo avaliativo foi, verificar se a acurácia atendia para efetuar a batalha, além de calcular a taxa de aprendizado e verificar a taxa de erro ao quadrado.

A ideia de utilizar uma taxa de aprendizado foi tirada do Perceptron onde é um classificador linear isso quer dizer que ele só irá lidar com problemas de classificação onde o conjunto de dados seja linearmente separável.



Conforme colocado o DataSet é totalmente regular e estável, nenhuma das variáveis irá mudar durante o processo para um valor desconhecido, por este motivo a ideia de colocar uma taxa de aprendizado para classificar de forma linear é interessante.

A operação é bem simples. Mas a ideia é saber como o perceptron chega nos valores ideais de cada peso fazendo assim com que cada instância seja classificada de forma correta.

Justificativa

Os valores foram bastante satisfatórios, onde foi criado um campo de batalha randômico e executado o processo de predição destes dados, segue trecho do código:

```
# Avaliando vencedores
import random

combats_name = test_data[cols].replace(pokemon.Name)

# Recupero o e faço um random para impressão dos dados dos pokémons
print(random.randint(0, len(combats_name)))

combats_name[0:random.randint(0,total)]
```

E gerou as seguintes saídas:

	First_pokemon	Second_pokemon	Winner
0	Staryu	Koffing	Staryu
1	Klink	Espeon	Espeon
2	Reshiram	Hitmonchan	Reshiram
3	Ampharos	Dwebble	Dwebble
4	Fearow	Joltik	Fearow
5	Seadra	Gligar	Gligar
6	Monferno	Sunflora	Monferno
7	Chansey	Nidoqueen	Chansey
8	Cyndaquil	Gothorita	Cyndaquil
9	Pelipper	Rufflet	Rufflet
10	Nidoking	Armaldo	Armaldo
11	Blastoise	Dustox	Dustox
12	Scizor	Meowstic Female	Scizor
13	Pelipper	Clawitzer	Clawitzer
14	Shieldon	Monferno	Monferno
15	Feebas	Metapod	Metapod
16	Aipom	Kyurem	Kyurem
17	Slowbro	Duosion	Duosion
18	Seaking	Mienshao	Mienshao
19	Lillipup	Pansage	Pansage
20	Riolu	Shinx	Shinx
21	Nidoqueen	Cradily	Nidoqueen
22	Magmortar	Poochyena	Magmortar
23	Shellder	Claydol	Shellder
24	Vullaby	Heatran	Heatran

Dessa forma entende-se que o processo de Random Forest foi executado de forma consistente.

V. Conclusão

Foi visto que um problema bem básico que pode ser resolvido usando o Machine Learning. Os conceitos abordados neste trabalho formam a base da maioria das abordagens de Machine Learning. A explicação dos conceitos da maneira mais simples, para que possa ter uma compreensão justa de como o Machine Learning funciona e pode ser usado no mundo real.

Forma livre de visualização

Foi verificado que é importante ter um bom DataSet de entrada para que os dados e a dinâmica das suas escolhas seja facilitada e bem executada.

Reflexão

Foi definitivamente descrito tudo que ocorreu no projeto.

Um aspecto importante foi entender que nem todos os cenários é necessário utilizar Deep Learning e ou cálculos de tensores, com um modelo mais simples consegue predizer de forma consistente o trabalho e as formas de execução.

Houve alguns momentos difíceis, como versões de bibliotecas, escolher bem o momento da normalização dos dados, todo esse processo gerou-se momentos de pesquisa nos materiais das aulas anteriores.

Melhorias

Como qualquer outro trabalho é necessário várias melhorias importantes, exemplo: executar todo o processo do jogo, melhorar a acurácia do algoritmo implementado e dentre outros processos que são importante para a melhoria contínua.